
How do Kaspersky's
advanced algorithms
ensure the best
protection for your
business against
cyber threats?

Machine Learning and Human Expertise

Contents

Our Classical Approach to Automatic Detection	2
Heuristics-Based Approach to Automatic Detection	3
Behavior Detection	4
Sandboxes, KSN and... People	4
Smart Records	5
Anti-Virus Records Based on Decision Trees	5
Locality-Sensitive Hashing	6
The Malware Path	7
Keeping False Positives at Bay	8

Machine Learning and Human Expertise

Every year, the quality of our protection wins Kaspersky more awards than anyone else in the cybersecurity industry. This achievement would be impossible without the fusion of a global big data 'cyberbrain' powered by machine learning algorithms and the unequalled expertise of our security teams in combating 'next-gen' threats. We offer you a 'sneak peek' into the heart of Kaspersky's anti-malware infrastructure, revealing our algorithms and their role in fighting the most dangerous threats to businesses like yours.

«How do Kaspersky's advanced algorithms ensure the best protection for your business against cyber threats?»

Our Classical Approach to Automatic Detection

Our virus collection contains samples of detectable threats grouped by detection names, e.g. Backdoor.Win32.Hupigon.abc. When a new, undetected sample arrives, we begin by searching our collection for similar samples. The search principle is roughly the same as that used by Google Search. The only difference is that Google Search is word-based, while our searches are based on file features. In the simplest scenario, if the sample has been unpacked successfully we can extract the strings responsible for the malware functionality and use them in much the same way as keywords are used by a search engine.

At Kaspersky we have an automated system that handles both the analysis of files and the automatic classification of threats.




Image size:
330 × 200


Find other sizes of this image:
All sizes - Small - Medium

Best guess for this image: **Internet bot**

Internet bot - Wikipedia
https://en.wikipedia.org/wiki/Internet_bot ▼
An internet bot, also known as web robot, WWW robot or simply bot, is a software application that runs automated tasks (scripts) over the Internet. Typically, bots ...

What is an Internet Bot? - Definition from Techopedia
<https://www.techopedia.com/definition/24063/internet-bot> ▼
An Internet bot, in its most generic sense, is software that performs an automated task over the Internet. More specifically, a bot is an automated application used ...

Visually similar images



Report images

Pages that include matching images

Loki Bot: On a hunt for corporate passwords | Securelist
<https://securelist.com/Spam-and-phishing> ▼
1200 × 732 - Aug 29, 2018 - The messages discovered so far contain an attachment with an iso extension that Kaspersky Lab solutions detect as Loki Bot. The malware's ...

Google service that searches for similar pictures on the Internet.

This system sorts the inbound stream of samples while simultaneously adding hashes to identify and define detections. One simple hash record covers the detection of just one file, but in this way we can be sure there will be no 'false positives'.

When malware for which the collection contains no similar samples turns up, we know that this is either something completely new, or that it's not malware at all. This is where the expertise of human AV Analysts comes into play. By unpacking and detecting a sample, the analyst creates a sort of "center of gravity" in the collection. Over time, other modified versions of the new sample will automatically gravitate towards this reference point.

Heuristics-Based Approach to Automatic Detection

Exclusively hash-based detection only gets you so far: one slight file modification (e.g. a single byte added at the end), and the whole file becomes undetectable again. That's why we unleash our heuristics-based automatic detection system on the whole family of our malware samples (like Backdoor.Win32.Hubigon.abc etc). With the help of an emulator, the heuristics-based system creates execution logs of all the samples, finds their common execution patterns and creates a single execution-based heuristic record. The benefit of this approach is that new malware samples exhibiting similar behavior will be detected, even if the content includes some changes.

Let's take a closer look at the process by which heuristic detection records are created. The robotic system uses machine learning to extract key execution sequences. The machine doesn't know or care what particular purpose any sequence of commands serves. As far as it's concerned, it's enough to know that this or that execution sequence – or combination of sequences – is characteristic of some malware family and could not occur in any clean file. After some iterations, the most effective indicators and their combinations are automatically consolidated into records.

Unlike this robot, an experienced human analyst can understand exactly what the sample is up to, despite any attempts to shake the heuristic system's emulator off its trail. So he or she can write a record straight away, highlighting obvious malware-like behavior.

These two different approaches tend to work in parallel, particularly when automatic detection results are inconclusive and an expert second opinion is needed. Robotic and human-made records then work in tandem, ensuring successful detection.

To evade detection, the malefactor may change the functionality of his or her malware. But there are limitations. Let's assume the malware has basic functionality: downloading a file via a malicious link, saving the file to disk and starting it (Trojan-Downloader). There are no more than 10 programmatic ways to download anything from the internet, and no more than five ways to start an executable file. When the malefactor has tried them all and found that every method is detected, their best option is probably to give up and instead mount an attack against a business with no security solution, or a solution lacking execution analysis tools.

```
KERNEL32!LoadLibrary(0x004020B6 "KERNEL32.dll");
KERNEL32!GetTickCount();
KERNEL32!LoadLibrary(0x00403000 "kernel32.dll");
KERNEL32!LoadLibrary(0x0040302C "urlmon.dll");
urlmon!URLDownloadToFile(,0x00403061 "http [REDACTED]",0x004030C5 "c
KERNEL32!Sleep()
KERNEL32!DeleteFile(0x004030C5 "c:\\boot.bak");
urlmon!URLDownloadToFile(,0x0040308F "http [REDACTED]",0x004030B9 "c:\\4
```

Trojan-Downloader.Win32.Small.aon execution log

The malefactor may have another trick up their sleeve: knowing the emulation specifics, they can try to disrupt the emulation process by, for example, inserting long execution delays or asking for system parameters the emulator can't provide. While some of these tricks can themselves be treated as indicators for detection, we can nonetheless detect the sample's true functionality through a further method – using System Watcher, a system that monitors a process's activities within the actual operating system.

Behavior Detection

Unlike the emulator, this component is a true behavioral detection system based on logs of real-life sample executions, so it's impossible to fool. It has its own set of behavioral records, which in many ways resemble those of the emulator-based detection system.

The scope of logging performed by Behavior Detection is considerably wider than is possible during emulation. And, unlike the latter process, this logging has unlimited timeframes: everything suspicious encountered within a given context is considered and cached until enough evidence for detection has been gathered. If malicious activity is detected, the action is simply rolled back.

As with the emulation system, Behavior Detection has its role in both on-premise detection and as a part of our in-lab wizardry. Incidentally, Behavior Detection activity is transparent and has no adverse impact on the process being monitored.

Continuous on-premise behavioral analysis creates an extremely powerful detection layer, but unleashing the power of Kaspersky infrastructure to execute suspicious files, study their behavior and detect threats via a fast response threat input database such as KSN (Kaspersky Security Network) is even more effective.

Sandboxes, KSN and... People

We continuously test samples – both known malicious and unknown – in our internal behavioral Sandbox systems. Some of these Sandboxes mimic user systems running standard products, while the most powerful have tremendously granular logging capabilities, allowing extremely fine-tuned detection.

The Sandbox logs, along with Behavior Detection execution statistics received from KSN's voluntary participants, are processed by both robots and human experts. Robots run two important processes: the logs of new malicious samples' execution are studied using Machine Learning to find new detection indicators – and unknown samples are also detected, with static records created for subsequent use both in our infrastructure and on customers' premises. So even if malware creators are resourceful enough to sidestep the majority of the on-premise detection layers – usually through extensive reconnaissance and preliminary testing – they'll be no better off in the end.

Meanwhile, using robot-distilled indicators, human experts create effective behavioral records similar to those based on emulated execution, but with a very much wider range of indicators to utilize.



Smart Records

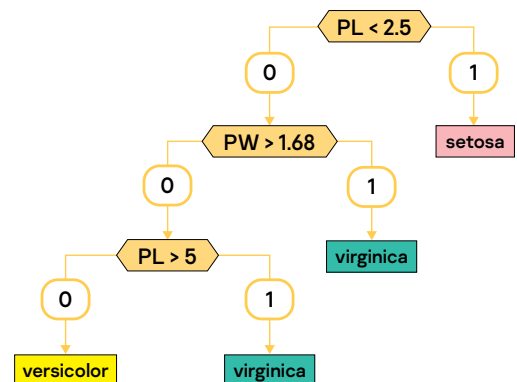
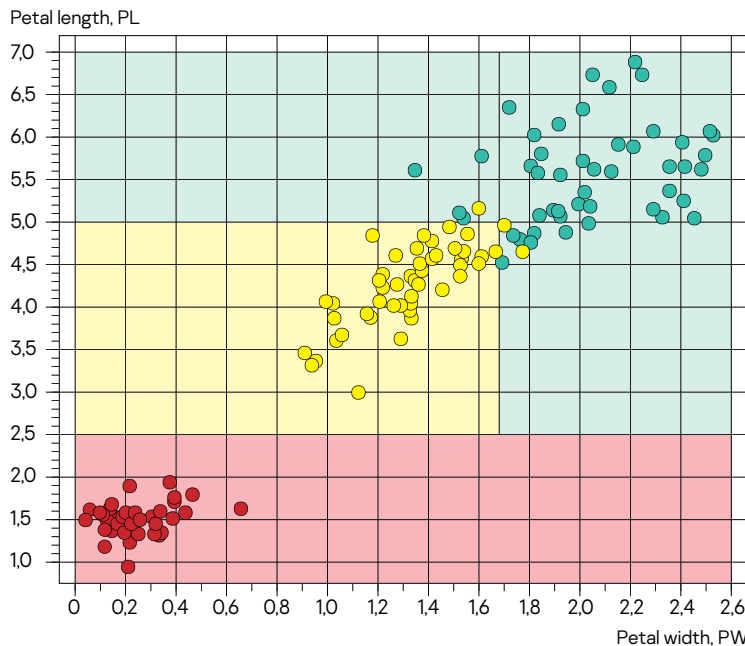
The list of Machine learning-based processes doesn't end with the above. There are further robotic detection layers capable of detecting sizable families of malware. Usually we call these 'Smart Records'.

Anti-Virus Records Based on Decision Trees

The in-lab robotic part of this system analyzes the same collection of samples as above, and creates or improves records based on decision trees. This makes it possible to separate files into classes and to specify criteria sensitive to the features of those files.

How does this work? Let's look at an example based on the Iris flower data set, a typical test case for statistical classification techniques. Say we have 150 flowers: 50 samples each of Iris Setosa, Iris Virginica and Iris Versicolor. To simplify the task, let's take the two most informative features of these flowers: petal length (PL) and petal width (PW). Plotting the features of each sample provides data that can be used to create a decision tree which can then allocate one of the three classes to each new Iris sample through 'request-response', like this:

Our AV engine uses exactly the same kind of tree. Each decision tree is carefully fine-tuned and delivered to the user. The selected features of an individual file running on the user's computer are extracted and run through each decision tree. The tree then uses the responses to decide whether or not the file is malicious.



Axes of the 2 most informative attributes (of 4) two classes splitted accurately, 3 mistakes in 3rd class.
Source: Coursera/Yandex

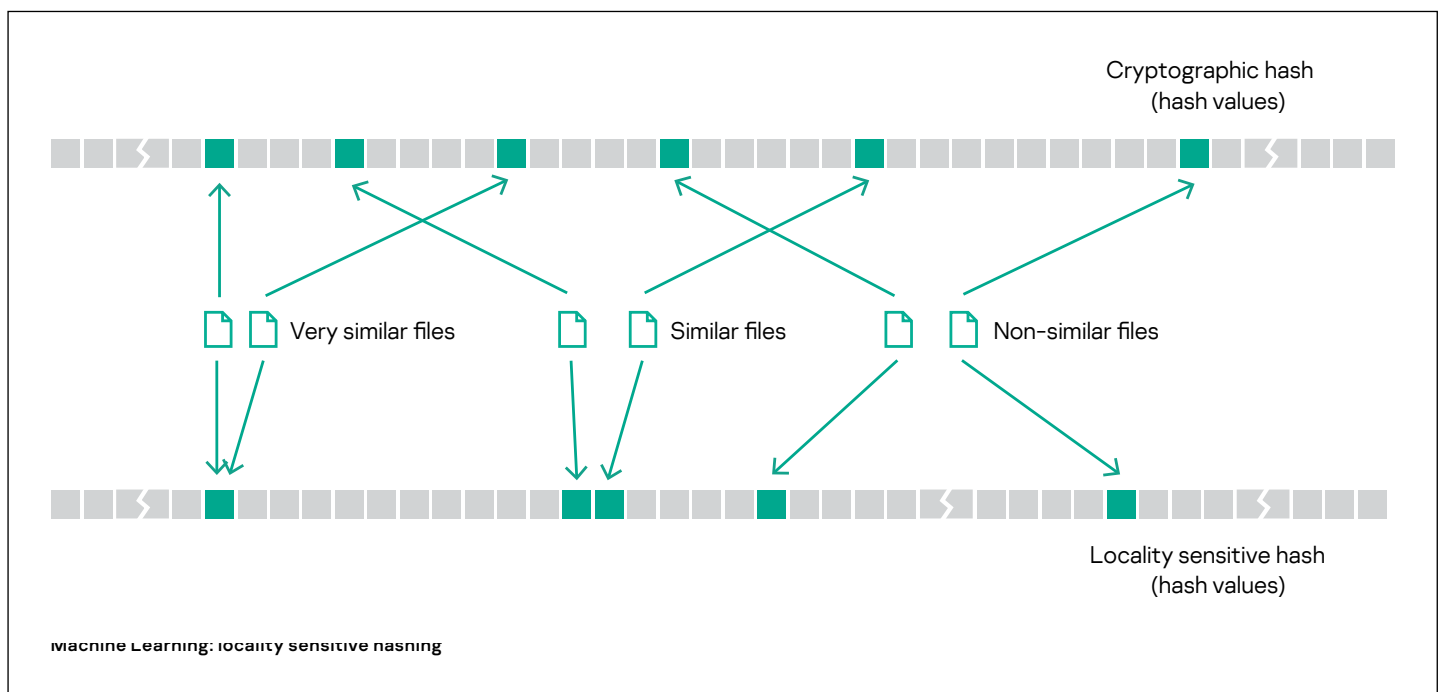
The advantage of this approach is its generalizing capabilities: each tree is created in our infrastructure based on a subset of the samples we have, but on users' computers the tree will also detect any samples not yet acquired by our researchers. For example, in the picture above, any dot in the red zone will be detected as Iris Setosa. A single tree-based record replaces an average of a thousand hash records.

Machine learning is indispensable to creating decision trees. While an expert can feed long lists of features to the robot, experts don't create tree-based records on their own. Only a machine can extract and apply the data, selecting the best features and, most importantly, creating decision rules based on these features. The human expert just monitors the result and controls the process.

Locality-Sensitive Hashing

Tree-based detection models are great but still have one major shortcoming: while being created automatically in our internal infrastructure, they can only operate effectively on the host (user's computer) where the particular file is studied. A cloud system based on this principle would create considerable network traffic, which is undesirable in most cases.

Hash-based cloud systems, by contrast, are considerably lighter in terms of traffic. But a typical cryptographic hash, such as MD5 or SHA256 for example, nearly always corresponds to one file only. It's good that you won't find a second file with the same hash; false positives are out of the question here. But it would be great having a hash that's the same for all malware belonging to the same family. In other words, insignificant file modifications would not affect the hash. This is in fact possible with so-called LSH, or Locality-Sensitive Hashing. Requests resulting in detections based on this hash can be made via the cloud.



How do we calculate levels of similarity between files? Consider the following example.

Assume that **File A** is characterized by the following numerical features:
31, 83, 98, 86, 183, 79, 67, 153, 77, 67

Meanwhile, **File B** is slightly different:
27, 89, 93, 81, 190, 71, 67, 161, 75, 69

All numbers can be "rounded down" by dividing them by 10. We get:

File A: 3, 8, 9, 8, 18, 7, 6, 15, 7, 6

File B: 2, 8, 9, 8, 19, 7, 6, 16, 7, 6

As you can see, the feature values are almost identical now.

Here's another approach: calculate the arithmetic mean of numbers in the first and second halves of each of the two files above. The answer turns out to be:

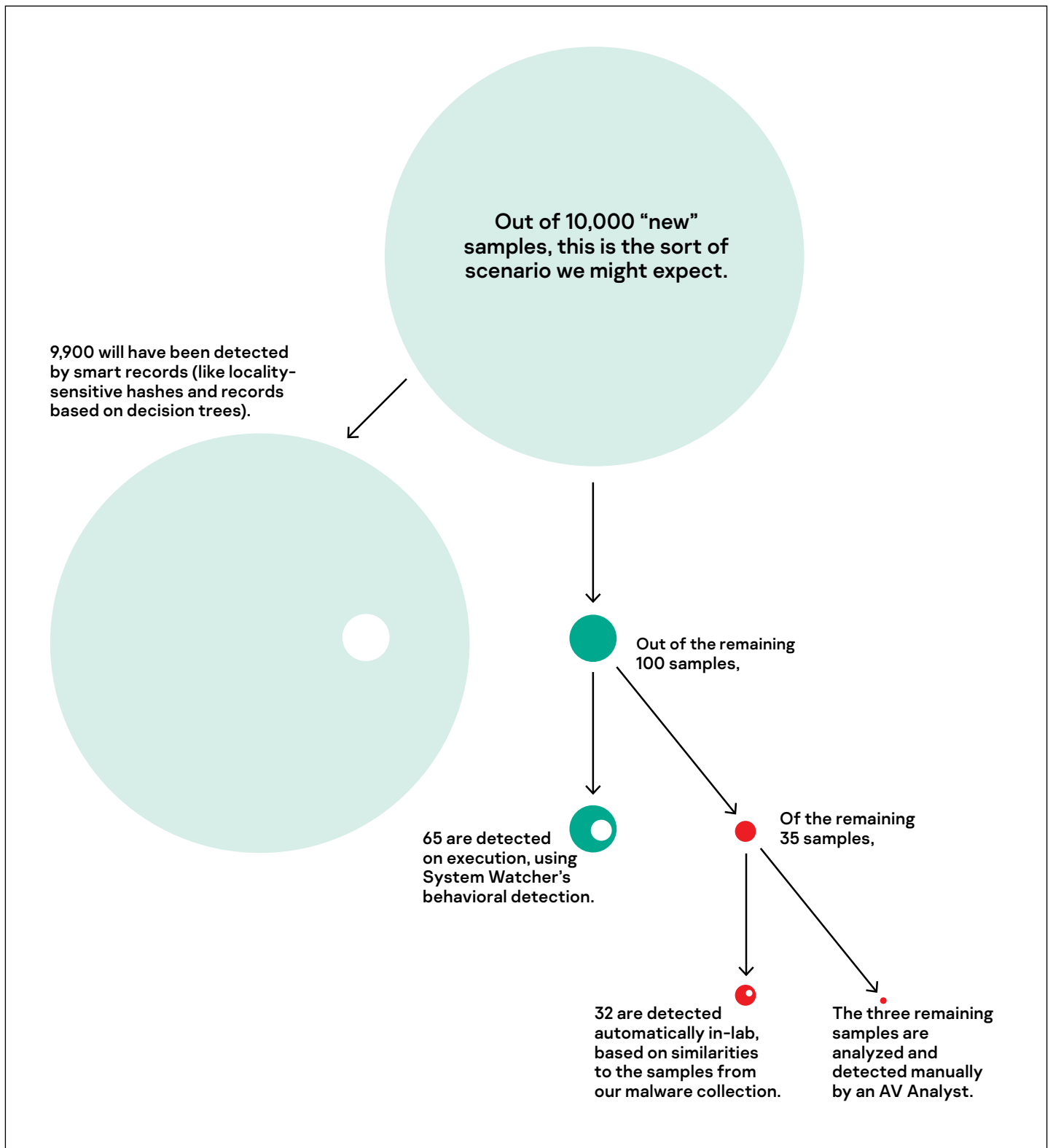
File A: 96, 88

File B: 96, 88

In this case, the LSH hashes are identical.

The challenging aspect of this approach involves choosing features that vary slightly within the same malware family, but which are still different enough to be recognized where a specific clean file is concerned. These features then need to be "quantized" (simply put, they are processed so that their precision is reduced). As you may have guessed, only a robot can do this. But the task is still formulated by a human expert.

The Malware Path



All samples (regardless of how they made it into the collection) are frequently re-analyzed for any new detections using generalization technologies (previously described heuristic autorecords/tree-based records/locality-sensitive hash records). If a sample was previously detected using only the individual hash, the detection is “generalized” by machine learning, so it’s included into some big ‘family’ of malware described only by a single record. After that, the individual hash record is deleted.

Keeping False Positives at Bay

The story of Machine Learning-powered heuristic detection would not be complete without mentioning the issue of false positives. As with any method based on the generalization principle, these techniques contain the inherent potential for mistakes, resulting in false positive detections. Unexpected shifts in the threat landscape can increase the probability of this happening, so, as well as the continuous constant adjustment of detection models, constant and very tight control over false positives is required.

Kaspersky products incorporate automated mechanisms for the tracking, timely switch-off and correction of faulty records. But, following the principle of multi-layeredness in everything and the best possible outcome for customers, all the records, including those created by robots, are under constant scrutiny from the most experienced analysts. They make sure that the records are thoroughly tested and adjusted at appropriate intervals, to ensure the highest possible detection rates while keeping the number of false positives as close to zero as possible. As independent tests consistently prove, they are extremely good at this!

All the technologies and approaches described here are instrumental in achieving True Cybersecurity – but we're always inventing new ones, to keep every next generation of threats at bay.

Enterprise Cybersecurity: www.kaspersky.com/enterprise
TechnoWiki: www.kaspersky.com/technowiki
IT Security News: www.kaspersky.com/blog
Cyber Threats News: www.securelist.com

www.kaspersky.com

kaspersky